실험 5. Queue 활용

1. 목적

이 실험에서는 실시간 운영체제가 제공하는 멀티태스킹 환경에서, 스캔 방식의 키 매트릭스를 구동하는 경우, 키 매트릭스를 스캔하는 태스크를 분리하고 Keyboard 관련한 함수들을 API로 분리하여 응용 프로그램을 작성하는 실험이다.

2. 실험 내용 설명

이 실험은 실험 4의 모든 것을 그대로 사용하되, 키보드 관련 처리를 별도의 태스크로 분리하고, 키 매트릭스에서 만들어진 키 입력과 관련된 기능을 다음과 같은 3개의 함수를 통해 추상화하여 응용 프로그램에서 이용하는 실험이다.

```
void key_init(void);      // Keyboard Queue를 초기화 (Queue를 비움)
int kbhit(void);          // Queue에 입력된 키 데이터가 있으면 1을, 없으면 0을 리턴
u8 getkey(void);          // Queue에 키 데이터가 있으면 그 키 코드 (0x00 ~ 0x0F)를 리턴
                          // 없으면, 키를 누를 때 까지 대기
```

이런 Queue 시스템을 구현하기위해서는 RTOS가 지원하는 Queue Primitive를 이용할 필요가 있다. http://www.freertos.org/의 <API Reference> - <Queue> 섹션을 참고하면 된다. 앞 홈페이지에는 각 API에 대한 설명과 예제가 나와있다.

주요 사용 API는 xQueueCreate(), xQueueSend(), xQueueReceive(), xQueuePeek() 이다.

이 함수들을 이용하려면 “queue.h″를 #include 해야 한다.

(1) xQueueHandle **xQueueCreate**(
unsigned portBASE_TYPE uxQueueLength,
unsigned portBASE_TYPE uxItemSize
);

Creates a new queue instance. This allocates the storage required by the new queue and returns a handle for the queue.

Parameters:

uxQueueLength The maximum number of items that the queue can contain.

uxItemSize The number of bytes each item in the queue will require. Items are queued by copy, not by reference, so this is the number of bytes that will be copied for each posted item. Each item on the queue must be the same size.

Returns:
If the queue is successfully create then a handle to the newly created queue is returned. If the queue cannot be created then 0 is returned.

(2) portBASE_TYPE **xQueueSend**(
xQueueHandle xQueue,
const void * pvItemToQueue,
portTickType xTicksToWait
);

Post an item on a queue. The item is queued by copy, not by reference.

Parameters:
xQueue The handle to the queue on which the item is to be posted.
pvItemToQueue A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from pvItemToQueue into the queue storage area.
xTicksToWait The maximum amount of time the task should block waiting for space to become available on the queue, should it already be full. The call will return immediately if the queue is full and xTicksToWait is set to 0. The time is defined in tick periods so the constant portTICK_RATE_MS should be used to convert to real time if this is required.
If INCLUDE_vTaskSuspend is set to '1' then specifying the block time as portMAX_DELAY will cause the task to block indefinitely (without a timeout).

Returns:
pdTRUE if the item was successfully posted, otherwise errQUEUE_FULL.

(3) portBASE_TYPE **xQueueReceive**(
xQueueHandle xQueue,
void *pvBuffer,
portTickType xTicksToWait
);

Receive an item from a queue. The item is received by copy so a buffer of adequate size must be provided. The number of bytes copied into the buffer was defined when the queue was created.

Parameters:
pxQueue The handle to the queue from which the item is to be received.
pvBuffer Pointer to the buffer into which the received item will be copied.
xTicksToWait The maximum amount of time the task should block waiting for an item to receive should the queue be empty at the time of the call. Setting xTicksToWait to 0 will cause the function to return immediately if the queue is empty. The time is defined in tick periods so the constant portTICK_RATE_MS should be used to convert to real time if this is required.
If INCLUDE_vTaskSuspend is set to '1' then specifying the block time as portMAX_DELAY will cause the task to block indefinitely (without a timeout).

Returns:
pdTRUE if an item was successfully received from the queue, otherwise pdFALSE.

(4) portBASE_TYPE **xQueuePeek**(
xQueueHandle xQueue,
void *pvBuffer,
portTickType xTicksToWait
);

Receive an item from a queue without removing the item from the queue. The item is received by copy so a buffer of adequate size must be provided. The number of bytes copied into the buffer was defined when the queue was created.

Successfully received items remain on the queue so will be returned again by the next call, or a call to xQueueReceive().

Parameters:
xQueue The handle to the queue from which the item is to be received.
pvBuffer Pointer to the buffer into which the received item will be copied. This must be at least large enough to hold the size of the queue item defined when the queue was created.

xTicksToWait The maximum amount of time the task should block waiting for an item to receive should the queue be empty at the time of the call. The time is defined in tick periods so the constant portTICK_RATE_MS should be used to convert to real time if this is required.
If INCLUDE_vTaskSuspend is set to '1' then specifying the block time as portMAX_DELAY will cause the task to block indefinitely (without a timeout).

Returns:
pdTRUE if an item was successfully received (peeked) from the queue, otherwise pdFALSE.

3. 예제 소스

다음 소스는 main.c의 소스 일부이며, Keyboard Queue를 선언하여(line 13), Queue를 생성하며(line 25), Key Matrix를 스캔하기 위한 Key_Task 함수를 Task로 선언한 것(line 11, 18)이다. 26번 라인에는 어떤 이유든 Queue가 생성되지 않은 경우에 에러 처리를 해야 하지만 생략되었다.

```
10 static portTASK_FUNCTION(Exp_Task, pvParameters);
11 portTASK_FUNCTION(Key_Task, pvParameters);
12
13 xQueueHandle KeyQueue;
14
15 int
16 main(void)
17 {
18     xTaskCreate(Key_Task,
19                         (const signed char * const)"Key_Task",
20                         2048,
21                         (void *)NULL,
22                         tskIDLE_PRIORITY + 10,
23                         NULL);
24
25     KeyQueue = xQueueCreate(MAX_KEY_LOG, sizeof(u8));
26     // Error Processing Needed !
```

다음 예제 소스는 Exp_Homework.c Template에 있는 소스의 일부로서 위에서 정한 key API들을 구현한 부분이다.

```
void
key_init(void)
{
   int i;
   u8 key;

   for (i = 0; i < MAX_KEY_LOG; i++)
      xQueueReceive(KeyQueue, &key, 0);
}

int
kbhit(void)
{
   u8 key;
   int ret = xQueuePeek(KeyQueue, &key, 0);
   return (ret == pdPASS);
}

u8
getkey(void)
{
   u8 key;
   xQueueReceive(KeyQueue, &key, portMAX_DELAY);
   return key;
}
```

key_init() 함수는 이미 Queue에 있는 것을 모두 읽어서 버리는 코드이며, kbhit() 함수는 xQueuePeek()을 이용하여 Queue에서 꺼내지는 않고 읽을 것이 있는지를 확인하여 읽을 것이 있으면, Key Data가 있는 것이므로 1을 리턴한다. getkey()는 xQueueReceive()를 뭔가 읽을 때까지 (마지막 인자가 portMAX_DELAY) 대기하도록 구현되어 있다.

4. 응용 실험 (과제)

이번 과제는 API 방식의 Key Matrix 구동이 정상적으로 동작하는 것을 확인하기 위한 실험이다. 소스 Template의 Exp_Homework.c의 Key_Task() 함수와 Exp_5_Homework_A(), Exp_5_Homework_B(), 함수를 작성한다. Key_Task() 함수는 별도의 Task로서, KeyMatrix를 scan하여 앞서 선언된 KeyQueue에 xQueueSend()를 이용하여 눌린 Key (0x00 ~ 0x0F)를 넣도록 작성된다.

4.1 응용 실험 A.

지난주의 Key Matrix 실험의 일부를 구현하는데 이번에 새로 만든 API 적용해보는 실험이다. 즉, 키를 누르는 순서대로 7 Segment LED에 누른 키를 제일 오른쪽에 차례로 표시하고, 이전에 누른 키는 왼쪽 방향으로 스크롤 해 나간다. 즉, 7 Segment LED의 제일 왼쪽에는 가

장 최근에 누른 키 값이, 제일 오른쪽에서 7번째 전에 눌렀던 키 값이 표시되는 프로그램을 작성하는 것이다.

* 이 프로그램을 구현할 때는 kbhit() 함수는 사용하지 말고, getkey() 만을 이용한다. getkey()안에서 대기를 하므로, 이전의 while 문들에서와 같이 Start 버튼을 확인할 수 없다. 따라서, Key Matrix에서 '0' 키가 눌리면, 응용 실험 B로 넘어가도록 한다.

4.2 응용 실험 B.

BARLED1의 LED를 0.5초 마다 왼쪽부터 오른쪽으로, 오른쪽 끝에 이르면 다시 왼쪽으로 왔다갔다 하게 만들면서 위 응용 실험 A를 수행한다. 이번엔 kbhit() 함수도 사용하여야 하며, 응용 실험 B는 '0' 키도 스크롤 대상이 되며, Nintendo DS의 'Start' 버튼으로 실험을 끝내도록 한다. BARLED1을 켜는 동작과 7 Segment LED를 움직이는 동작은 상관없는 동작이기 때문에 서로 다른 태스크로 만드는 것이 당연하지만 kbhit()에 의한 polling을 구현한다는 실험 목적상 한 함수 내에서 구현한다.

*** 주의 사항 : Key Matrix에 대한 반응이 빠르게 만들어야 함 !!!!!!!!!!

5-1. 소스 및 샘플 프로그램

A. Template 소스는 자료실의 KeyQueue-Homework.zip에 있음
B. 미리 만들어 놓은 Binary는 자료실의 KeyQueue.nds 임

5-2. 소스 및 샘플 프로그램 (Github 이용)

소스는 Github의 KeyQueue 소스를 check-out하여 과제를 수행한다.

https://github.com/hl1itj/nds-ide/tree/microprocessor/lab/KeyQueue

(이 문서와 미리 build한 binary KeyQueue.nds는 doc directory에 있음)

Checkout 한 뒤,
https://github.com/hl1itj/nds-ide/tree/microprocessor/학번/KeyQueue

에서 각자 개발을 진행한다.